DOI:
10.1109/DCC.2019.00062

Document Version
Peer reviewed version

Constructing Antidictionaries in Output-Sensitive Space

Lorraine A.K. Ayad*, Golnaz Badkobeh†, Gabriele Fici◊, Alice Héliou§ and Solon P. Pissis‡

*Department of Informatics
King's College London, London, UK
lorraine.ayad@kcl.ac.uk

†Department of Computing
Goldsmiths University of London, London, UK
g.badkobeh@gold.ac.uk

◊Dipartimento di Matematica e Informatica
Università di Palermo, Palermo, Italy
gabriele.fici@unipa.it

§Independent Researcher
alice.heliou@gmail.com

‡CWI, Amsterdam, The Netherlands
solon.pissis@cwi.nl

Abstract

A word x that is absent from a word y is called *minimal* if all its proper factors occur in y. Given a collection of k words $y_1, y_2, \ldots, y_k$ over an alphabet Σ, we are asked to compute the set $\mathrm{M}^{\ell}_{y_1\#\ldots\#y_k}$ of minimal absent words of length at most ℓ of word $y = y_1\#y_2\#\ldots\#y_k$, $\# \notin \Sigma$. In data compression, this corresponds to computing the antidictionary of k documents. In bioinformatics, it corresponds to computing words that are absent from a genome of k chromosomes. This computation generally requires $\Omega(n)$ space for $n = |y|$ using any of the plenty available $\mathcal{O}(n)$-time algorithms. This is because an $\Omega(n)$-sized text index is constructed over y which can be impractical for large n. We do the identical computation incrementally using output-sensitive space. This goal is reasonable when $||\,\mathrm{M}^{\ell}_{y_1\#\ldots\#y_N}\,|| = o(n)$, for all $N \in [1,k]$. For instance, in the human genome, $n \approx 3 \times 10^9$ but $||\,\mathrm{M}^{12}_{y_1\#\ldots\#y_k}\,|| \approx 10^6$. We consider a constant-sized alphabet for stating our results. We show that *all* $\mathrm{M}^{\ell}_{y_1}, \ldots, \mathrm{M}^{\ell}_{y_1\#\ldots\#y_k}$ can be computed in $\mathcal{O}(kn + \sum_{N=1}^{k} ||\,\mathrm{M}^{\ell}_{y_1\#\ldots\#y_N}\,||)$ total time using $\mathcal{O}(\textsc{MaxIn} + \textsc{MaxOut})$ space, where $\textsc{MaxIn}$ is the length of the longest word in $\{y_1, \ldots, y_k\}$ and $\textsc{MaxOut} = \max\{||\,\mathrm{M}^{\ell}_{y_1\#\ldots\#y_N}\,|| : N \in [1,k]\}$. Proof-of-concept experimental results are also provided confirming our theoretical findings and justifying our contribution.

1 Introduction

The word x is an *absent word* of the word y if it does not occur in y. The absent word x of y is called *minimal* if and only if all its proper factors occur in y. The set of all minimal absent words for a word y is denoted by M_y. The set of all minimal absent words of length at most ℓ of a word y is denoted by M^{ℓ}_y. For example, if $y = \mathtt{abaab}$, then $\mathrm{M}_y = \{\mathtt{aaa}, \mathtt{aaba}, \mathtt{bab}, \mathtt{bb}\}$ and $\mathrm{M}^{3}_y = \{\mathtt{aaa}, \mathtt{bab}, \mathtt{bb}\}$. The upper bound on the number of minimal absent words is $\mathcal{O}(\sigma n)$ [1], where σ is the size of the alphabet and n is the length of y, and this is tight for integer alphabets [2]; in fact, for large alphabets, such as when $\sigma \geq \sqrt{n}$, this bound is also tight even for minimal absent words having the same length [3].

State-of-the-art algorithms compute all minimal absent words of y in $\mathcal{O}(\sigma n)$ time [1, 4] or in $\mathcal{O}(n + |\,\mathrm{M}_y\,|)$ time [5, 6] for integer alphabets. There also exist space-efficient data structures based on the Burrows-Wheeler transform of y that can be applied for this computation [7, 8]. In many real-world applications of minimal absent words, such as in data compression [9, 10, 11, 12], in sequence comparison [2, 6], in on-line pattern matching [13], or in identifying pathogen-specific signatures [14], only a subset of minimal absent words may be considered, and, in particular, the minimal absent words of length (at most) ℓ. Since, in

the worst case, the number of minimal absent words of y is $\Theta(\sigma n)$, $\Omega(\sigma n)$ space is required to represent them explicitly. In [6], the authors presented an $\mathcal{O}(n)$-sized data structure for outputting minimal absent words of a specific length in optimal time for integer alphabets.

The problem with existing algorithms for computing minimal absent words is that they make use of $\Omega(n)$ space; and the same amount is required even if one is merely interested in the minimal absent words of length at most ℓ. This is because all of these algorithms construct global data structures, such as the suffix array [4]. In theory, this problem can be addressed by using the external memory algorithm for computing minimal absent words presented in [15]. The I/O-optimal version of this algorithm, however, requires a lot of external memory to build the global data structures for the input [16]. One could also use the algorithm of [17] that computes M_y^ℓ in $\mathcal{O}(n + |\,\mathrm{M}_y^\ell\,|)$ time using $\mathcal{O}(\min\{n, \ell z\})$ space, where z is the size of the LZ77 factorisation of y. This algorithm also requires constructing the truncated DAWG, a type of global data structure which could take space $\Omega(n)$. Thus, in this paper, we investigate whether M_y^ℓ can be computed efficiently in output-sensitive space. As y can be "decomposed" into a collection of k words—with a suitable overlap of length ℓ so as not to lose information—we consider the following, general, computational problem.

Problem Given k words $y_1, y_2, \ldots, y_k$ over an alphabet Σ and an integer $\ell > 0$, compute the set $\mathrm{M}^\ell_{y_1\#\ldots\#y_k}$ of minimal absent words of length at most ℓ of $y = y_1\#y_2\#\ldots\#y_k$, $\# \notin \Sigma$.

In data compression, this scenario corresponds to computing the antidictionary of k documents [9, 10]. In bioinformatics, it corresponds to computing words that are absent from a genome of k chromosomes. As discussed above, this computation generally requires $\Omega(n)$ space for $n = |y|$. We do the identical computation incrementally using output-sensitive space. This goal is reasonable when $||\,\mathrm{M}^\ell_{y_1\#\ldots\#y_N}\,|| = o(n)$, for all $N \in [1, k]$. In the human genome, $n \approx 3 \times 10^9$ but $||\,\mathrm{M}^{12}_{y_1\#\ldots\#y_k}\,|| \approx 10^6$, where k is the total number of chromosomes.

Our Results Antidictionary-based compressors work on $\Sigma = \{\mathtt{0}, \mathtt{1}\}$ and in bioinformatics we have $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$; we thus consider a constant-sized alphabet for stating our results. We show that *all* $\mathrm{M}^\ell_{y_1}, \ldots, \mathrm{M}^\ell_{y_1\#\ldots\#y_k}$ can be computed in $\mathcal{O}(kn + \sum_{N=1}^{k} ||\,\mathrm{M}^\ell_{y_1\#\ldots\#y_N}\,||)$ total time using $\mathcal{O}(\textsc{MaxIn} + \textsc{MaxOut})$ space, where $\textsc{MaxIn}$ is the length of the longest word in $\{y_1, \ldots, y_k\}$ and $\textsc{MaxOut} = \max\{||\,\mathrm{M}^\ell_{y_1\#\ldots\#y_N}\,|| : N \in [1, k]\}$. Proof-of-concept experimental results are provided confirming our theoretical findings and justifying our contribution.

2 Preliminaries

We generally follow [18]. An *alphabet* Σ is a finite ordered non-empty set of elements called *letters*. A *word* is a sequence of elements of Σ. The set of all words over Σ of length at most ℓ is denoted by $\Sigma^{\leq \ell}$. We fix a constant-sized alphabet Σ, i.e., $|\Sigma| = \mathcal{O}(1)$. Given a word $y = uxv$ over Σ, we say that u is a *prefix* of y, x is a *factor* (or subword) of y, and v is a *suffix* of y. We also say that y is a *superword* of x. A factor x of y is called *proper* if $x \neq y$.

Given a word y over Σ, the set of *minimal absent words* (MAWs) of y is defined as

$$\begin{aligned}\mathrm{M}_y =&\{aub \mid a,b \in \Sigma, au \text{ and } ub \text{ are factors of } y \text{ but } aub \text{ is not}\}\\ &\cup \{c \in \Sigma \mid c \text{ does not occur in } y\}.\end{aligned}$$

For instance, over $\Sigma = \{\mathtt{a},\mathtt{b},\mathtt{c}\}$, for $y = \mathtt{ab}$ we have $\mathrm{M}_y = \{\mathtt{aa},\mathtt{bb},\mathtt{ba},\mathtt{c}\}$. MAWs of length 1 for y can be found in $\mathcal{O}(|y| + |\Sigma|) = \mathcal{O}(|y|)$ time using $\mathcal{O}(|\Sigma|) = \mathcal{O}(1)$ working space, and so, in what follows, we focus on the computation of MAWs of length at least 2.

The *suffix tree* $\mathcal{T}(y)$ of a non-empty word y of length n is the compact trie representing all suffixes of y [18]. The *branching* nodes of the trie as well as the *terminal* nodes, that correspond to non-empty suffixes of y, become *explicit* nodes of the suffix tree, while the other nodes are *implicit*. We let $\mathcal{L}(v)$ denote the *path-label* from the root node to node v. We say that node v is path-labeled $\mathcal{L}(v)$; i.e., the concatenation of the edge labels along the path from the root node to v. Additionally, $\mathcal{D}(v) = |\mathcal{L}(v)|$ is used to denote the *word-depth* of node v. A node v such that the path-label $\mathcal{L}(v) = y[i\,..\,n-1]$, for some $0 \leq i \leq n-1$, is *terminal* and is also labeled with index i. Each factor of y is uniquely represented by either an explicit or an implicit node of $\mathcal{T}(y)$ called its *locus*. The *suffix-link* of a node v with path-label $\mathcal{L}(v) = aw$ is a pointer to the node path-labeled w, where $a \in \Sigma$ is a single letter and w is a word. The suffix-link of v exists by construction if v is a non-root branching node of $\mathcal{T}(y)$. The *matching statistics* of a word $x[0\,..\,|x|-1]$ with respect to word y is an array $\mathrm{MS}_x[0\,..\,|x|-1]$, where $\mathrm{MS}_x[i]$ is a pair (f_i, p_i) such that (i) $x[i\,..\,i+f_i-1]$ is the longest prefix of $x[i\,..\,|x|-1]$ that is a factor of y; and (ii) $y[p_i\,..\,p_i+f_i-1] = x[i\,..\,i+f_i-1]$ [19]. $\mathcal{T}(y)$ is constructible in time $\mathcal{O}(n)$, and, given $\mathcal{T}(y)$, we can compute MS_x in time $\mathcal{O}(|x|)$ [19].

3 Combinatorial Properties

For convenience, we consider the following setting. Let y_1, y_2 be words over the alphabet Σ and let $y_3 = y_1 \# y_2$, with $\# \notin \Sigma$. Let ℓ be a positive integer and set $\mathrm{M}^\ell_{y_1} = \mathrm{M}_{y_1} \cap \Sigma^{\leq \ell}$ and $\mathrm{M}^\ell_{y_2} = \mathrm{M}_{y_2} \cap \Sigma^{\leq \ell}$. We want to construct $\mathrm{M}^\ell_{y_3} = \mathrm{M}_{y_3} \cap \Sigma^{\leq \ell}$. Let $x \in \mathrm{M}^\ell_{y_3}$. We have two cases:

Case 1 : $x \in \mathrm{M}^\ell_{y_1} \cup \mathrm{M}^\ell_{y_2}$;

Case 2 : $x \notin \mathrm{M}^\ell_{y_1} \cup \mathrm{M}^\ell_{y_2}$.

The following auxiliary fact follows directly from the minimality property.

Fact 1. *Word x is absent from word y if and only if x is a superword of a MAW of y.*

For Case 1, we prove the following lemma.

Lemma 1 (Case 1)**.** *A word $x \in \mathrm{M}^\ell_{y_1}$ (resp. $x \in \mathrm{M}^\ell_{y_2}$) belongs to $\mathrm{M}^\ell_{y_3}$ if and only if x is a superword of a word in $\mathrm{M}^\ell_{y_2}$ (resp. in $\mathrm{M}^\ell_{y_1}$).*

Proof. Let $x \in \mathrm{M}^\ell_{y_1}$ (the case $x \in \mathrm{M}^\ell_{y_2}$ is symmetric). Suppose first that x is a superword of a word in $\mathrm{M}^\ell_{y_2}$, that is, there exists $v \in \mathrm{M}^\ell_{y_2}$ such that v is a factor of x. If $v = x$, then $x \in \mathrm{M}^\ell_{y_1} \cap \mathrm{M}^\ell_{y_2}$ and therefore, using the definition of MAW, $x \in \mathrm{M}^\ell_{y_3}$. If v is a proper factor of x, then x is an absent word of y_2 and again, by definition of MAW, $x \in \mathrm{M}^\ell_{y_3}$.

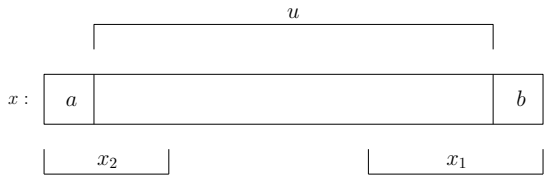


Figure 1: x_2 occurs in y_1 but not in y_2; x_1 occurs in y_2 but not in y_1; therefore aub does not occur in $y_1\#y_2$. By construction, au occurs in y_1 and ub occurs in y_2; therefore aub is a Case 2 MAW.

Suppose now that x is not a superword of any word in $\mathrm{M}^{\ell}_{y_2}$. Then x is not absent in y_2 by Fact 1, and hence in y_3, thus x cannot belong to $\mathrm{M}^{\ell}_{y_3}$. □

It should be clear that the statement of Lemma 1 implies, in particular, that all words in $\mathrm{M}^{\ell}_{y_1} \cap \mathrm{M}^{\ell}_{y_2}$ belong to $\mathrm{M}^{\ell}_{y_3}$. Furthermore, Lemma 1 motivates us to introduce the *reduced set of MAWs* of y_1 with respect to y_2 as the set $\mathrm{R}^{\ell}_{y_1}$ obtained from $\mathrm{M}^{\ell}_{y_1}$ after removing those words that are superwords of words in $\mathrm{M}^{\ell}_{y_2}$. The set $\mathrm{R}^{\ell}_{y_2}$ is defined analogously.

Example 1. Let $y_1 = \mathtt{abaab}$, $y_2 = \mathtt{bbaaab}$ and $\ell = 5$. We have $\mathrm{M}^{\ell}_{y_1} = \{\mathtt{bb,aaa,bab,aaba}\}$ and $\mathrm{M}^{\ell}_{y_2} = \{\mathtt{bbb,aaaa,baab,aba,bab,abb}\}$. The word $\mathtt{bab}$ is contained in $\mathrm{M}^{\ell}_{y_1} \cap \mathrm{M}^{\ell}_{y_2}$ so it belongs to $\mathrm{M}^{\ell}_{y_3}$. The word $\mathtt{aaba} \in \mathrm{M}^{\ell}_{y_1}$ is a superword of $\mathtt{aba} \in \mathrm{M}^{\ell}_{y_2}$ hence $\mathtt{aaba} \in \mathrm{M}^{\ell}_{y_3}$. On the other hand, the words $\mathtt{bbb}$, $\mathtt{aaaa}$ and $\mathtt{abb}$ are superwords of words in $\mathrm{M}^{\ell}_{y_1}$, hence they belong to $\mathrm{M}^{\ell}_{y_3}$. The remaining MAWs are not superwords of MAWs of the other word. The reduced sets are therefore $\mathrm{R}^{\ell}_{y_1} = \{\mathtt{bb}, \mathtt{aaa}\}$ and $\mathrm{R}^{\ell}_{y_2} = \{\mathtt{baab}, \mathtt{aba}\}$. In conclusion, we have for Case 1 that $\mathrm{M}^{\ell}_{y_3} \cap (\mathrm{M}^{\ell}_{y_1} \cup \mathrm{M}^{\ell}_{y_2}) = \{\mathtt{aaaa,bab,aaba,abb,bbb}\}$. □

We now investigate the set $\mathrm{M}^{\ell}_{y_3} \setminus (\mathrm{M}^{\ell}_{y_1} \cup \mathrm{M}^{\ell}_{y_2})$ (Case 2).

Fact 2. *Let $x = aub$, $a, b \in \Sigma$, be such that $x \in \mathrm{M}^{\ell}_{y_3}$ and $x \notin \mathrm{M}^{\ell}_{y_1} \cup \mathrm{M}^{\ell}_{y_2}$. Then au occurs in y_1 but not in y_2 and ub occurs in y_2 but not in y_1, or vice versa.*

The rationale for generating the reduced sets should become clear with the next lemma.

Lemma 2 (Case 2)**.** *Let $x \in \mathrm{M}^{\ell}_{y_3} \setminus (\mathrm{M}^{\ell}_{y_1} \cup \mathrm{M}^{\ell}_{y_2})$. Then x has a prefix x_i in $\mathrm{R}^{\ell}_{y_i}$ and a suffix x_j in $\mathrm{R}^{\ell}_{y_j}$, for i, j such that $\{i, j\} = \{1, 2\}$.*

Proof. Let $x = aub$, $a, b \in \Sigma$, be a word in $\mathrm{M}^{\ell}_{y_3} \setminus (\mathrm{M}^{\ell}_{y_1} \cup \mathrm{M}^{\ell}_{y_2})$. By Fact 2, au occurs in y_1 but not in y_2 and ub occurs in y_2 but not in y_1, or vice versa. Let us assume the first case holds (the other case is symmetric). Since au does not occur in y_2, there is a MAW $x_2 \in \mathrm{M}^{\ell}_{y_2}$ that is a factor of au. Since ub occurs in y_2, x_2 is not a factor of ub. Consequently, x_2 is a prefix of au.

Analogously, there is an $x_1 \in \mathrm{M}^{\ell}_{y_1}$ that is a suffix of ub. Furthermore, x_1 and x_2 cannot be factors one of another. Inspect Figure 1 in this regard. □

Example 2. Let $y_1 = \mathtt{abaab}$, $y_2 = \mathtt{bbaaab}$ and $\ell = 5$. Consider $x = \mathtt{abaaa} \in \mathrm{M}^{\ell}_{y_3} \setminus (\mathrm{M}^{\ell}_{y_1} \cup \mathrm{M}^{\ell}_{y_2})$ (Case 2 MAW). We have that $\mathtt{abaa}$ occurs in y_1 but not in y_2 and $\mathtt{baaa}$ occurs in y_2 but not

in y_1. Since abaa does not occur in y_2, there is a MAW $x_2 \in \mathrm{R}^{\ell}_{y_2}$ that is a factor of abaa. Since baaa occurs in y_2, x_2 is not a factor of baaa. So x_2 is a prefix of abaa and this is aba. Analogously, there is MAW $x_1 \in \mathrm{R}^{\ell}_{y_1}$ that is a suffix of abaaa and this is aaa. □

As a consequence of Lemma 2, in order to construct the set $\mathrm{M}^{\ell}_{y_3} \setminus (\mathrm{M}^{\ell}_{y_1} \cup \mathrm{M}^{\ell}_{y_2})$, we should consider all pairs (x_i, x_j) with x_i in $\mathrm{R}^{\ell}_{y_i}$ and x_j in $\mathrm{R}^{\ell}_{y_j}$, $\{i, j\} = \{1, 2\}$. In order to construct the final set $\mathrm{M}^{\ell}_{y_1\#\ldots\#y_N}$, we use incrementally Lemmas 1 and 2. We summarise the whole approach in the following general theorem, which forms the theoretical basis of our technique.

Theorem 1. *Let $N > 1$, and let $x \in \mathrm{M}^{\ell}_{y_1\#\ldots\#y_N}$. Then, either $x \in \mathrm{M}^{\ell}_{y_1\#\ldots\#y_{N-1}} \cup \mathrm{M}^{\ell}_{y_N}$ (Case 1 MAWs) or, otherwise, $x \in \mathrm{M}^{\ell}_{y_i\#y_N} \setminus (\mathrm{M}^{\ell}_{y_i} \cup \mathrm{M}^{\ell}_{y_N})$ for some i. Moreover, in this latter case, x has a prefix in $\mathrm{R}^{\ell}_{y_1\#\ldots\#y_{N-1}}$ and a suffix in $\mathrm{R}^{\ell}_{y_N}$, or the converse, i.e., x has a prefix in $\mathrm{R}^{\ell}_{y_N}$ and a suffix in $\mathrm{R}^{\ell}_{y_1\#\ldots\#y_{N-1}}$ (Case 2 MAWs).*

Proof. Let $x \in \mathrm{M}^{\ell}_{y_1\#\ldots\#y_N}$ and $x \notin \mathrm{M}^{\ell}_{y_1\#\ldots\#y_{N-1}} \cup \mathrm{M}^{\ell}_{y_N}$. Then, $x \notin \mathrm{M}^{\ell}_{y_1\#\ldots\#y_{N-1}}$ and $x \notin \mathrm{M}^{\ell}_{y_N}$. Let x be a word of length m. By the definition of MAW, $x[0\,..\,m-2]$ and $x[1\,..\,m-1]$ must both be factors of $y_1\#\ldots\#y_N$. However, both cannot be factors of $y_1\#\ldots\#y_{N-1}$ and both cannot be factors of y_N. Therefore, we have one of the two cases:

Case 1 : $x[0\,..\,m-2]$ is factor of $y_1\#\ldots\#y_{N-1}$ but not of y_N and $x[1\,..\,m-1]$ is a factor of y_N but not of $y_1\#\ldots\#y_{N-1}$.

Case 2 : $x[0\,..\,m-2]$ is factor of y_N but not of $y_1\#\ldots\#y_{N-1}$ and $x[1\,..\,m-1]$ is a factor of $y_1\#\ldots\#y_{N-1}$ but not of y_N.

These two cases are symmetric, thus only proof of Case 1 will be presented here. If $x[0]$ does not occur in y_N then $x[0] \in \mathrm{R}^{\ell}_{y_N}$. Otherwise, let $x[0\,..\,t]$ be the longest prefix of $x[0\,..\,m-2]$ that is a factor of y_N.

Because $0 \le t < m-1$ then $x[1\,..\,t+1]$ is a factor of y_N. Therefore, $x[0\,..\,t+1] \in \mathrm{M}^{\ell}_{y_N}$. In addition, all factors of $x[0\,..\,t+1]$ occur in $y_1\#\ldots\#y_{N-1}$, so $x[0\,..\,t+1] \in \mathrm{R}^{\ell}_{y_N}$.

Now, $x[1\,..\,m-1]$ does not occur in $y_1\#\ldots\#y_{N-1}$, so either $x[m-1]$ does not occur in $y_1\#\ldots\#y_{N-1}$ which means that $x[m-1] \in \mathrm{R}^{\ell}_{y_1\#\ldots\#y_{N-1}}$, or let $x[p\,..\,m-1]$ be the longest suffix of $x[1\,..\,m-1]$ that occurs in $y_1\#\ldots\#y_{N-1}$.

Because $0 < p \le m-1$ then $x[p-1\,..\,m-2]$ occurs in $y_1\#\ldots\#y_{N-1}$, therefore $x[p-1\,..\,m-1] \in \mathrm{M}^{\ell}_{y_1\#\ldots\#y_{N-1}}$. Since all factors of $x[p-1\,..\,m-1]$ occur in y_N, we have $x[p-1\,..\,m-1] \in \mathrm{R}^{\ell}_{y_1\#\ldots\#y_{N-1}}$. □

4 Algorithm

Let us first introduce an algorithmic tool. In the *weighted ancestor* problem, introduced in [20], we consider a rooted tree T with an integer weight function μ defined on the nodes. We require that the weight of the root is zero and the weight of any other node is strictly larger than the weight of its parent. A weighted ancestor query, given a node v and an integer value $w \le \mu(v)$, asks for the highest ancestor u of v such that $\mu(u) \ge w$, i.e., such an ancestor u that $\mu(u) \ge w$ and $\mu(u)$ is the smallest possible. When T is the suffix tree of a word y of length n, we can locate the locus of any factor $y[i\,..\,j]$ using a weighted ancestor

query. We define the weight of a node of the suffix tree as the length of the word it represents. Thus a weighted ancestor query can be used for the terminal node decorated with i to create (if necessary) and mark the node that corresponds to $y[i\,..\,j]$.

Theorem 2 ([21]). *Given a collection Q of weighted ancestor queries on a weighted tree T on n nodes with integer weights up to $n^{\mathcal{O}(1)}$, all the queries in Q can be answered* off-line *in $\mathcal{O}(n + |Q|)$ time.*

4.1 The Algorithm

At the Nth step, we have in memory the set $\mathrm{M}^{\ell}_{y_1\#\ldots\#y_{N-1}}$. Our algorithm works as follows:

1. We read word y_N from the disk and compute $\mathrm{M}^{\ell}_{y_N}$ in time $\mathcal{O}(|y_N|)$. We output the words in the following constant-space form: $< i_1, i_2, \alpha >$ per word [4]; such that $y_N[i_1\,..\,i_2] \cdot \alpha \in \mathrm{M}^{\ell}_{y_N}$.
2. Here we compute Case 1 MAWs. We apply Lemma 1 to construct set $M = \{w : w \in \mathrm{M}^{\ell}_{y_1\#\ldots\#y_N}, w \in \mathrm{M}^{\ell}_{y_1\#\ldots\#y_{N-1}} \cup \mathrm{M}^{\ell}_{y_N}\}$ and the sets $\mathrm{R}^{\ell}_{y_1\#\ldots\#y_{N-1}}, \mathrm{R}^{\ell}_{y_N}$ as follows.
   (a) We first want to find the elements of $\mathrm{M}^{\ell}_{y_1\#\ldots\#y_{N-1}}$ that are superwords of any word $y_N[i_1\,..\,i_2]\cdot\alpha$. We build the generalised suffix tree $T_1 = \mathcal{T}(\mathrm{M}^{\ell}_{y_1\#\ldots\#y_{N-1}} \cup \{y_N\})$ [19]. We find the locus of the longest proper prefix $y_N[i_1\,..\,i_2]$ of each element of $\mathrm{M}^{\ell}_{y_N}$ in T_1 via answering off-line a batch of weighted ancestor queries (Theorem 2). From there on, we spell α and mark the corresponding node on T_1, if any. After processing all $< i_1, i_2, \alpha >$ in the same manner, we traverse T_1 to gather all occurrences (starting positions) of words $y_N[i_1\,..\,i_2] \cdot \alpha$ in the elements of $\mathrm{M}^{\ell}_{y_1\#\ldots\#y_{N-1}}$, thus finding the elements of $\mathrm{M}^{\ell}_{y_1\#\ldots\#y_{N-1}}$ that are superwords of any $y_N[i_1\,..\,i_2]\cdot\alpha$. By definition, no MAW $y_N[i_1\,..\,i_2] \cdot \alpha$ is a prefix of another MAW $y_N[i'_1\,..\,i'_2] \cdot \alpha'$, thus the marked nodes form pairwise disjoint subtrees, and the whole process takes time $\mathcal{O}(|y_N| + ||\,\mathrm{M}^{\ell}_{y_1\#\ldots\#y_{N-1}}\,||)$, the size of T_1.
   (b) We next want to check if the words $y_N[i_1\,..\,i_2] \cdot \alpha$ are superwords of any element of $\mathrm{M}^{\ell}_{y_1\#\ldots\#y_{N-1}}$. We first sort all tuples $< i_1, i_2, \alpha >$ using radixsort and then check this using the matching statistics algorithm for y_N with respect to $\mathcal{T}(\mathrm{M}^{\ell}_{y_1\#\ldots\#y_{N-1}})$ considering the tuples in ascending order (from left to right) at the same time. By definition, no element in $\mathrm{M}^{\ell}_{y_1\#\ldots\#y_{N-1}}$ is a factor of another element in the same set. Thus if a factor of $y_N[i_1\,..\,i_2] \cdot \alpha$ corresponds to an element in $\mathrm{M}^{\ell}_{y_1\#\ldots\#y_{N-1}}$ this is easily located in $\mathcal{T}(\mathrm{M}^{\ell}_{y_1\#\ldots\#y_{N-1}})$ while running the matching statistics algorithm. The whole process takes $\mathcal{O}(|y_N|+||\,\mathrm{M}^{\ell}_{y_1\#\ldots\#y_{N-1}}\,||)$ time: $\mathcal{O}(||\,\mathrm{M}^{\ell}_{y_1\#\ldots\#y_{N-1}}\,||)$ time to construct the suffix tree and a further $\mathcal{O}(|y_N|)$ time for the matching statistics algorithm and for processing the $\mathcal{O}(|y_N|)$ tuples.

   We create set $\mathrm{R}^{\ell}_{y_1\#\ldots\#y_{N-1}}$ explicitly since it is a subset of $\mathrm{M}^{\ell}_{y_1\#\ldots\#y_{N-1}}$. We create set $\mathrm{R}^{\ell}_{y_N}$ implicitly: every element $x \in \mathrm{R}^{\ell}_{y_N}$ is stored as a tuple $< i_1, i_2, \alpha >$ such that $x = y_N[i_1\,..\,i_2] \cdot \alpha$. We store every element of $\{x_2 : x_2 \in M \cap \mathrm{M}^{\ell}_{y_N}\}$ with the same representation. All other elements of M are stored explicitly.

3. Construct the suffix tree of y_N and use it to locate all occurrences of words in $\mathrm{R}^{\ell}_{y_1\#\ldots\#y_{N-1}}$ in y_N and store the occurrences as pairs (starting position, ending position). This step can be done in time $\mathcal{O}(|y_N| + ||\,\mathrm{R}^{\ell}_{y_1\#\ldots\#y_{N-1}}\,||)$. By definition, no element in $\mathrm{R}^{\ell}_{y_1\#\ldots\#y_{N-1}}$ is a prefix of another element in $\mathrm{R}^{\ell}_{y_1\#\ldots\#y_{N-1}}$, and thus this can be done within the claimed time complexity.

4. For every $i \in [1, N-1]$, we perform the following to compute Case 2 MAWs:

   (a) Read word y_i from the disk. Construct the suffix tree T_x of word $x = y_i\#y_N$ in time $\mathcal{O}(|y_i| + |y_N|)$. Use T_x to locate all occurrences of elements of $\mathrm{R}^{\ell}_{y_N}$ in y_i and store the occurrences as pairs (starting position, ending position). This step can be done in time $\mathcal{O}(|y_i| + |y_N|)$ similar to step 2. By definition, no element in $\mathrm{R}^{\ell}_{y_N}$ is a prefix of another element in $\mathrm{R}^{\ell}_{y_N}$, and thus this can be done within the claimed time complexity.

   (b) During a bottom-up traversal of T_x mark, at each explicit node of T_x, the smallest starting position of the subword represented by that node, and the largest starting position of the same subword. This can be done in time $\mathcal{O}(|y_i| + |y_N|)$ by propagating upwards the labels of the terminal nodes (starting positions of suffixes) and updating the smallest and largest positions analogously.

   (c) Compute the set $\mathrm{M}^{\ell}_{y_i\#y_N}$ and output the words in the following constant-space form: $< a, i_1, i_2, b >$ per word; such that $a \cdot x[i_1\,..\,i_2] \cdot b$ is a MAW. This can be done in time $\mathcal{O}(|y_i| + |y_N|)$.

   (d) For each element of $\mathrm{M}^{\ell}_{y_i\#y_N}$, we need to locate the node representing word $ax[i_1\,..\,i_2] = au$ and the node representing word $x[i_1\,..\,i_2]b = ub$. This can be done in time $\mathcal{O}(|y_i| + |y_N|)$ via answering off-line a batch of weighted ancestor queries (Theorem 2). At this point, we have located the two nodes on T_x. We assign a pointer from the stored starting position g of au to the ending position f of ub, only if g is before # and f is after # (f can be trivially computed using the stored starting position of ub and the length of ub). Conversely, we assign a pointer from the ending position f of ub to the stored starting position g of au, only if f is before # and g is after #.

   (e) Suppose au occurs in y_i and ub in y_N. We make use of the pointers as follows. Recall steps 3 and 4(a) and check whether au starts where a word r_1 of $\mathrm{R}^{\ell}_{y_N}$ starts and ub ends where a word r_2 of $\mathrm{R}^{\ell}_{y_1\#\ldots\#y_{N-1}}$ ends. If this is the case and $|u| \geq \max\{|r_1|, |r_2|\} - 1$, then by Theorem 1 aub is added to our output set M, otherwise discard it. Inspect Figure 2 in this regard. Conversely, if au occurs in y_N and ub in y_i check whether au starts where a word r_2 of $\mathrm{R}^{\ell}_{y_1\#\ldots\#y_{N-1}}$ starts and whether ub ends where a word r_1 of $\mathrm{R}^{\ell}_{y_N}$ ends. If this is the case and $|u| \geq \max\{|r_1|, |r_2|\} - 1$, then aub is added to M, otherwise discard it.

Finally, we set $\mathrm{M}^{\ell}_{y_1\#\ldots\#y_N} = M$ as the output of the Nth step. Let MaxIn be the length of the longest word in $\{y_1, \ldots, y_k\}$ and MaxOut $= \max\{||\,\mathrm{M}^{\ell}_{y_1\#\ldots\#y_N}\,|| : N \in [1, k]\}$.

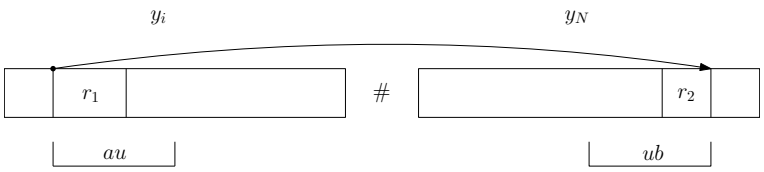


Figure 2: au starts where a word r_1 of $\mathrm{R}^{\ell}_{y_N}$ starts in y_i and ub ends where a word r_2 of $\mathrm{R}^{\ell}_{y_1\#\ldots\#y_{N-1}}$ ends in y_N. Moreover, if $|u| \geq \max\{|r_1|, |r_2|\} - 1$, then aub is a Case 2 MAW.

Theorem 3. *Given k words $y_1, y_2, \ldots, y_k$ and an integer $\ell > 0$, all $\mathrm{M}^{\ell}_{y_1}, \ldots, \mathrm{M}^{\ell}_{y_1\#\ldots\#y_k}$ can be computed in $\mathcal{O}(kn + \sum_{N=1}^{k} ||\, \mathrm{M}^{\ell}_{y_1\#\ldots\#y_N}\, ||)$ total time using $\mathcal{O}(\textsc{MaxIn} + \textsc{MaxOut})$ space, where $n = |y_1\# \ldots \#y_k|$.*

Proof. From the above discussion, the time is bounded by $\mathcal{O}(\sum_{N=1}^{k}\sum_{i=1}^{N-1}(|y_N| + |y_i|) + \sum_{N=1}^{k} ||\, \mathrm{M}^{\ell}_{y_1\#\ldots\#y_N}\, ||)$. We can bound the first term as follows.

$$\sum_{N=1}^{k}\sum_{i=1}^{N-1}(|y_N| + |y_i|) \leq \sum_{N=1}^{k}\sum_{i=1}^{k}(|y_N| + |y_i|) = \sum_{N=1}^{k}\sum_{i=1}^{k}|y_N| + \sum_{N=1}^{k}\sum_{i=1}^{k}|y_i| = 2k(|y_1| + \cdots + |y_k|).$$

Therefore the time is bounded by $\mathcal{O}(kn + \sum_{N=1}^{k} ||\, \mathrm{M}^{\ell}_{y_1\#\ldots\#y_N}\, ||)$.

The space is bounded by the maximum time spent at a single step; namely, the length of the longest word in the collection plus the maximum total size of set elements across all output sets. Note that the total output size of the algorithm is the sum of all its output sets, that is $\sum_{N=1}^{k} ||\, \mathrm{M}^{\ell}_{y_1\#\ldots\#y_N}\, ||$, and $\textsc{MaxOut}$ could come from any intermediate set.

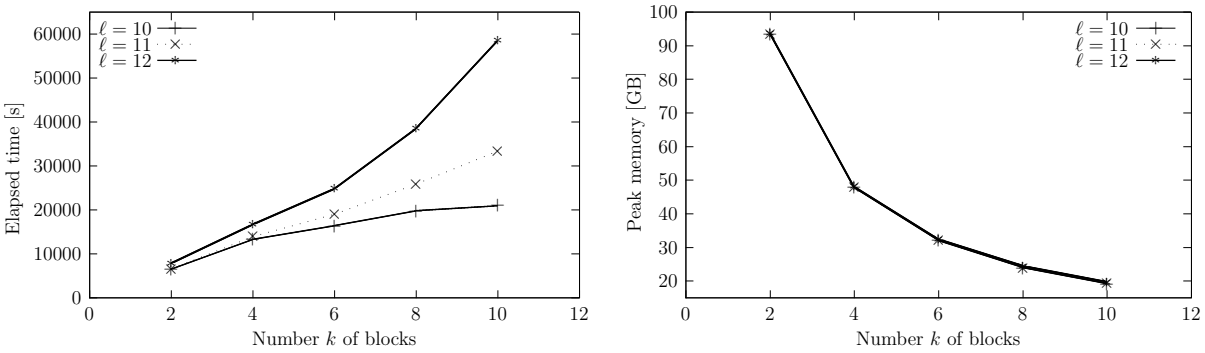
The correctness of the algorithm follows from Lemma 1 and Theorem 1. □

5 Proof-of-Concept Experiments

In this section, we do not directly compare against the fastest internal [4] or external [15] memory implementations because the former assumes that we have the required amount of internal memory, and the latter assumes that we have the required amount of external memory to construct and store the global data structures for a given input dataset. If the memory for constructing and storing the data structures is available, these linear-time algorithms are surely faster than the method proposed here. In what follows, we rather show that our output-sensitive technique offers a space-time tradeoff, which can be usefully exploited for specific values of ℓ, the maximal length of MAWs we wish to compute.

The algorithm discussed in Section 4 (with the exception of storing and searching the reduced set words explicitly rather than in the constant-space form previously described) has been implemented in the `C++` programming language[1]. The correctness of our implementation has been confirmed against that of [4]. As input dataset here we used the entire human genome (version hg38) [22], which has an approximate size of 3.1GB. The following experiment was conducted on a machine with an Intel Core i5-4690 CPU at 3.50 GHz and 128GB of memory running GNU/Linux. We ran the program by splitting the genome into

[1]The implementation can be made available upon request.



(a) **(b)**

Figure 3: Elapsed time and peak memory usage using increasing k blocks of the entire human genome for $\ell = 10, 11, 12$; notice that the peak memory usage is the same for all values of ℓ.

$k = 2, 4, 6, 8, 10$ blocks and setting $\ell = 10, 11, 12$. Figure 3 depicts the change in elapsed time and peak memory usage as k and ℓ increase (space-time tradeoff).

Graph (a) shows an increase of time as k and ℓ increase; and graph (b) shows a decrease in memory as k increases (as proved in Theorem 3). Notice that the space to construct the block-wise data structures bounds the total space used for the specific ℓ values and that is why the memory peak is essentially the same for the ℓ values used. This can specifically be seen for $\ell = 10$ where all words of length 10 are present in the genome. The same dataset was used to run the fastest internal memory implementation for computing MAWs [4] on the same machine. It took only 2242 seconds to compute all MAWs but with a peak memory usage of 60.80GB. The results confirm our theoretical findings and justify our contribution.

References

[1] Maxime Crochemore, Filippo Mignosi, and Antonio Restivo, "Automata and forbidden words," *Information Processing Letters*, vol. 67, no. 3, pp. 111–117, 1998.

[2] Panagiotis Charalampopoulos, Maxime Crochemore, Gabriele Fici, Robert Mercaş, and Solon P. Pissis, "Alignment-free sequence comparison using absent words," *Information and Computation*, vol. 262, no. 1, pp. 57–68, 2018.

[3] Yannis Almirantis, Panagiotis Charalampopoulos, Jia Gao, Costas S. Iliopoulos, Manal Mohamed, Solon P. Pissis, and Dimitris Polychronopoulos, "On avoided words, absent words, and their application to biological sequence analysis," *Algorithms for Molecular Biology*, vol. 12, no. 1, pp. 5:1–5:12, 2017.

[4] Carl Barton, Alice Héliou, Laurent Mouchard, and Solon P. Pissis, "Linear-time computation of minimal absent words using suffix array," *BMC Bioinformatics*, vol. 15, pp. 388, 2014.

[5] Yuta Fujishige, Yuki Tsujimaru, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda, "Computing DAWGs and minimal absent words in linear time for integer alphabets," in *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 - Kraków, Poland*, Piotr Faliszewski, Anca Muscholl, and Rolf Niedermeier, Eds. 2016, vol. 58 of *LIPIcs*, pp. 38:1–38:14, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

[6] Panagiotis Charalampopoulos, Maxime Crochemore, and Solon P. Pissis, "On extended special factors of a word," In Gagie et al. [23], pp. 131–138.

[7] Djamal Belazzougui, Fabio Cunial, Juha Kärkkäinen, and Veli Mäkinen, "Versatile succinct representations of the bidirectional Burrows-Wheeler transform," in *Algorithms - ESA 2013*

- 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings, Hans L. Bodlaender and Giuseppe F. Italiano, Eds. 2013, vol. 8125 of *Lecture Notes in Computer Science*, pp. 133–144, Springer.

[8] Djamal Belazzougui and Fabio Cunial, "A framework for space-efficient string kernels," *Algorithmica*, vol. 79, no. 3, pp. 857–883, 2017.

[9] Maxime Crochemore, Filippo Mignosi, Antonio Restivo, and Sergio Salemi, "Data compression using antidictionaries," *Proceedings of the IEEE*, vol. 88, no. 11, pp. 1756–1768, 2000.

[10] Maxime Crochemore and Gonzalo Navarro, "Improved antidictionary based compression," in *22nd International Conference of the Chilean Computer Science Society (SCCC 2002), 6-8 November 2002, Copiapo, Chile*, 2002, pp. 7–13.

[11] Martin Fiala and Jan Holub, "DCA using suffix arrays," in *2008 Data Compression Conference (DCC 2008), 25-27 March 2008, Snowbird, UT, USA*. 2008, p. 516, IEEE Computer Society.

[12] Takahiro Ota and Hiroyoshi Morita, "On the adaptive antidictionary code using minimal forbidden words with constant lengths," in *Proceedings of the International Symposium on Information Theory and its Applications, ISITA 2010, 17-20 October 2010, Taichung, Taiwan.* 2010, pp. 72–77, IEEE.

[13] Maxime Crochemore, Alice Héliou, Gregory Kucherov, Laurent Mouchard, Solon P. Pissis, and Yann Ramusat, "Minimal absent words in a sliding window and applications to on-line pattern matching," in *Fundamentals of Computation Theory - 21st International Symposium, FCT 2017, Bordeaux, France, September 11-13, 2017, Proceedings*, Ralf Klasing and Marc Zeitoun, Eds. 2017, vol. 10472 of *Lecture Notes in Computer Science*, pp. 164–176, Springer.

[14] Raquel M. Silva, Diogo Pratas, Luísa Castro, Armando J. Pinho, and Paulo Jorge S. G. Ferreira, "Three minimal sequences found in Ebola virus genomes and absent from human DNA," *Bioinformatics*, vol. 31, no. 15, pp. 2421–2425, 2015.

[15] Alice Héliou, Solon P. Pissis, and Simon J. Puglisi, "emMAW: computing minimal absent words in external memory," *Bioinformatics*, vol. 33, no. 17, pp. 2746–2749, 2017.

[16] Juha Kärkkäinen, Dominik Kempa, and Simon J. Puglisi, "Parallel external memory suffix sorting," in *Combinatorial Pattern Matching - 26th Annual Symposium, CPM 2015, Ischia Island, Italy, June 29 - July 1, 2015, Proceedings*, Ferdinando Cicalese, Ely Porat, and Ugo Vaccaro, Eds. 2015, vol. 9133 of *Lecture Notes in Computer Science*, pp. 329–342, Springer.

[17] Yuta Fujishige, Takuya Takagi, and Diptarama Hendrian, "Truncated DAWGs and their application to minimal absent word problem," In Gagie et al. [23], pp. 139–152.

[18] Maxime Crochemore, Christophe Hancart, and Thierry Lecroq, *Algorithms on strings*, Cambridge University Press, 2007.

[19] Dan Gusfield, *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*, Cambridge University Press, New York, NY, USA, 1997.

[20] Martin Farach and S. Muthukrishnan, "Perfect hashing for strings: Formalization and algorithms," in *Combinatorial Pattern Matching, 7th Annual Symposium, CPM 96, Laguna Beach, California, USA, June 10-12, 1996, Proceedings*, Daniel S. Hirschberg and Eugene W. Myers, Eds. 1996, vol. 1075 of *Lecture Notes in Computer Science*, pp. 130–140, Springer.

[21] Carl Barton, Tomasz Kociumaka, Chang Liu, Solon P. Pissis, and Jakub Radoszewski, "Indexing weighted sequences: Neat and efficient," *CoRR*, vol. abs/1704.07625, 2017.

[22] W. James Kent, Charles W. Sugnet, Terrence S. Furey, Krishna M. Roskin, Tom H. Pringle, Alan M. Zahler, Haussler, and David, "The human genome browser at UCSC," *Genome Research*, vol. 12, no. 6, pp. 996–1006, 2002.

[23] Travis Gagie, Alistair Moffat, Gonzalo Navarro, and Ernesto Cuadros-Vargas, Eds., *String Processing and Information Retrieval - 25th International Symposium, SPIRE 2018, Lima, Peru, October 9-11, 2018, Proceedings*, vol. 11147 of *Lecture Notes in Computer Science*. Springer, 2018.